CERTIFICATE OF MAIL: I hereby certify that this correspondence is being transmitted via FIRST CLASS MAIL to the Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on August 12, 2003.

Dated: Signature: Rebekah Everest
(Rebekah Everest)

#16

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent Application of:
Inderpal S. Bhandari, et al.

Application No.: 09/416,414 Group Art Unit: 3624

Filed: October 12, 1999 Examiner: Ella Colbert

For: METHOD AND APPARATUS FOR FINDING HIDDEN PATTERNS IN THE CONTEXT OF QUERYING APPLICATIONS

MS: Non-Fee Amendment
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

DECLARATION UNDER 37 C.F.R. § 131

Dear Sir:

WE, INDERPAL S. BHANDARI, RAJIV PRATAP, AND KRISHNAKUMAR RAMANUJAM DECLARE AND STATE THAT:

1. We are the co-inventors of the "Method and Apparatus for Finding Hidden Patterns in the Context of Querying Applications" described and claimed in the above-identified U.S. Patent Application ("the Present Application").

2. We were advised and therefore believe that the Present Application was filed as a continuation-in-part of U.S. Provisional Patent Application Serial No. 60/103,948 filed October 12, 1998 ("the Provisional Application").

3. The Present Application was assigned to Virtual Gold, Inc. by all of the inventors on October 12, 1999. Exhibit I is a copy of the Notice of Recordation of Assignment document evidencing the recordation of the assignment on October 12, 1999.

4. Well prior to June 15, 1998, the claimed invention shown and described in the provisional and present applications was conceived and at least two prototypes were built and successfully tested, in facilities of Virtual Gold, Inc. the assignee of the present application.

5. Several documents are provided with this Declaration further establishing that the described and claimed invention was invented, that is, both conceived and reduced to practice, well prior to June 15, 1998.

6. Exhibits A, B and C are documents written well prior to June 15, 1998 and pertain to the VirtualMiner™ prototype. The VirtualMiner™ prototype comprises two parts: Analyzer and Viewer. Exhibit A describes the software requirements of the VirtualMiner.™ Exhibit B describes the architecture of the Analyzer, which is a software component that sets up and prepares data for analysis using OLAP (on-line analytical processing) as well as data mining techniques. Exhibit C describes the architecture of the Viewer. The actual date has been redacted from these documents.

7. Exhibits D, E, F, G and H documents written well prior to June 15, 1998 and pertains to the ActiveTable prototype. Exhibit D describes the functional and operational elements of the ActiveTable Prototype. Exhibit E describes the viewer user interface of the ActiveTable Prototype. Exhibit F describes the design elements of the ActiveTable Prototype, and represents pages 4-8 and redacted page 9 of the "ActiveTable Design Document." Exhibits G and H describe an InterScope application of the ActiveTable Prototype. The actual date has been redacted from these documents.

8. We further declare that all statement made herein of our own knowledge are true and that all statements made on information and belief are believed to be true; and that these statements were made with the knowledge that willful, false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful, false statements may jeopardize the validity of the application or any patent issuing thereon.

Date: 8/8/03 Signed:________________

Inderpal S. BHANDARI

Date: ________________ Signed:________________

Rajiv PRATAP

Date: ________________ Signed:________________

Krishnakumar RAMANUJAM

VIR 201 (10001987)

8. We further declare that all statements made herein of our own knowledge are true and that all statements made on information and belief are believed to be true; and that these statements were made with the knowledge that willful, false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issuing thereon

Date: ____________ Signed: ____________
Inderpal S. BHANDARI

Date: Aug. 11th 2003 Signed: ____________
Rajiv PRATAP

Date: August 11, 2003 Signed: ____________
Krishnakumar RAMANUJAM

VirtualMiner Software Requirements Specification

1 Introduction

VirtualMiner is a software component that will enable users of various kinds of data to view, filter, query and mine their data easily. The target audience consists of "lay" users of data (as opposed to trained statisticians and data analysts). VirtualMiner consists of two parts, the Analyzer and the Viewer. Used in conjunction, the two components provide users with an easy and intuitive way of interacting with and discovering knowledge in their data.

In a multi-user scenario, such as an intranet/Internet, the Analyzer would typically be used by a data analyst/webmaster to prepare the raw data for analysis. Once this has been done, the end-user can then use the Viewer to view, filter and query the data, and view data mining results.

In a single-user scenario, the Analyzer and Viewer can be used by the same person to set up data for analysis, and view the results.

Both the Analyzer as well as the Viewer can be executed from within tools that developers as well as end-users are already familiar with, including web browsers and popular visual development tools and database and spreadsheet applications. Thus, there is no need for re-learning a complex user interface.

1.1 References

1. (AF Paper)

2 The VirtualMiner Analyzer

2.1 Introduction

The VirtualMiner Analyzer enables a designer to:

- Set up raw data to enable viewing, filtering and querying
- Set up one or more Data Mining analyses on the data (the Analyzer uses the Attribute Focusing algorithm for mining data [1])

2.2 Analyzer Execution

The VirtualMiner Analyzer should be a software component, capable of running within several environments, viz.

- Web browsers, such as Microsoft Internet Explorer and Netscape Navigator
- Visual Development tools that are capable of importing software components, such as Microsoft Visual C++, Microsoft Visual Basic, Microsoft Visual J++, Symantec Café, Borland C++, Sun Java WorkShop, Sun BeanBox, etc.

- Popular database utilities, such as Microsoft Access and Lotus Approach
- Popular spreadsheet utilities, such as Microsoft Excel, and Lotus 1-2-3

2.3 Inputs

The VirtualMiner Analyzer should be able to handle raw data in the following formats:

- ASCII Text Files
- Microsoft Excel and Lotus 1-2-3 Spreadsheets
- Local databases - Microsoft Access, Paradox, Dbase, Lotus Approach
- All RDBMS servers – Oracle, Sybase, Informix, SQLServer, Gupta, DB2
- Tables embedded in HTML files

2.4 Processing

The processing by the VirtualMiner Analyzer achieves the following purposes:

- Enables fast OLAP queries on the raw data
- Enables fast and easy Data Mining queries on the raw data

The Analyzer should read the raw data, and process it, using the following parameters provided by the user:

- Column(s) of interest for OLAP queries
- Data Mining Questions, each question consisting of :
  - Variable(s) of interest
  - Decision Variable
  - Numeric Attribute, if any
  - Cutoff

(The User Interface for collecting these parameters is described in Section 2.6)

2.5 Outputs

In order to achieve the aims of fast and easy querying on the raw data, it may be necessary for the Analyzer to convert the raw data into a format more amenable to fast querying. In such a situation, the processed data should be stored along with the raw data, and linked to it in some manner, so that when the user views the raw data, the results of processing are also immediately available (refer Section 3 for more details)

2.6 User Interface

The User Interface of the Analyzer consists of a wizard, which guides the designer through the process of setting up the parameters for processing the data. At each step, the designed is shown sample results, using the inputs he / she has already provided. (If no inputs have been provided to the Analyzer, sample results shown should be based on "dummy" inputs).

On each screen, there will be a "Tutorial" button. Pressing this button would provide a short, animated tutorial on using the screen. The tutorial would provide help on using the controls on the screen, as well as suggest possible alternatives to the choices that the designer has made. This should be achieved in an easy-going, friendly and fun-to-use manner.

The purpose of the interface is to collect the following information:

- Type of input data (local database, ASCII text file, etc.)
- Name of file / database which contains data
- Name of table in which data are stored (required if input is from database / HTML file)
- Column(s) of interest for OLAP queries
- Data Mining Questions, each question consisting of the following :
  - Variable(s) of interest
  - Decision Variable
  - Numeric Attribute, if any
  - Cutoff

The User Interface consists of the following screens:

2.6.1 Screen 1 : Welcome Screen

This screen is invoked when the user starts the Analyzer. It displays a suitable welcome message, informing the designer of the process he / she is about to go through. The screen contains the following controls:

1. *Cancel Button*

Pressing this button will cause the Analyzer to exit

2. *Next Button*

Pressing this button will cause Screen 1 to close, to be replaced by Screen 2

2.6.2 Screen 2 : Data Source Screen

This screen is invoked when the user presses the *Next* button on Screen 1, or the *Previous* button on Screen 3. It contains the following controls:

1. *ASCII Text File Option*

The designer can select this option if the raw data are contained in an ASCII text file

2. *HTML File Option*
The designer can select this option if the raw data are contained in an HTML file.

3. *Local Database Option*
The designer can select this option if the raw data are contained in a table on a local database (Microsoft Access / Lotus Approach). Note that "local" in this context implies a "non-server" database. The database file itself could be located on a fileserver on a network. However, it is accessed as if it were a local file, and not through a database server, such as Sybase or Oracle.

4. *Database server Option*
The designer can select this option if the raw data are contained in a table on a database server.

5. *Spreadsheet Option*
The designer can select this option if the raw data are contained in a Microsoft Excel / Lotus 1-2-3 spreadsheet

6. *Next Button*
This button will be enabled if the data source type has been selected. Pressing this button will cause Screen 2 to close. One of the following actions is taken :
   - If the designer selected the Database Server option, Screen 4 is displayed
   - For all other options, Screen 3 is displayed

7. *Previous Button*
Pressing this button will cause Screen 2 to close, to be replaced by Screen 1

8. *Cancel Button*
Pressing this button will cause the Analyzer to exit

9. *Tutorial Button*
Pressing this button will cause the tutorial for this screen to start.

2.6.3 Screen 3 : File Select Dialog Box

This is the standard system File Select Dialog Box. It is displayed if the user has selected any option other than the Database server option on Screen 2. Using this screen, the designer should be able to select the file where the data are stored. (Depending on the selection in Screen 2, she would be able to select an ASCII text file, an MS Access or Lotus Approach database, an MS Excel or Lotus 1-2-3 spreadsheet, or an HTML file).

Once the designer selects a file, and closes this dialog box, one of the following actions is taken:

- If the designer selected an ASCII text file, Screen 5 is displayed
- If the designer selected a local database or HTML file, Screen 6 is displayed

- If the designer selected a spreadsheet, Screen 7 is displayed

If the designer closes this dialog box without selecting a file, control returns to Screen 2.

2.6.4 Screen 4 : ODBC Login Screen

This screen is the standard system ODBC Login screen for selecting a data source from a database server. It is displayed if the designer selects a Database server as the data source in Screen 2. The designer will be able to select any ODBC source for the data.

Upon successful login, this screen will close, to be replaced by Screen 7.

2.6.5 Screen 5 : ASCII Text File Import Options Screen

This screen is displayed if the designer has selected an ASCII text file as the data source. The designer can specify import options for the text file using this screen. This screen has the following controls:

1. *Field Separator drop-down list*

This list contains possible options for the field separator character. The program should guess the most appropriate character, which should be selected by default.

2. *Text Delimiter drop-down list*

This list contains possible options for the text delimiter character. The program should guess the most appropriate character, which should be selected by default.

3. *Next Button*

Pressing this button causes Screen 5 to close, and Screen 7 to be displayed

4. *Previous Button*

Pressing thus button causes Screen 5 to close, and Screen 2 to be displayed.

5. *Cancel Button*

Pressing this button causes the Analyzer to exit.

6. *Tutorial Button*

Pressing this button causes the tutorial for this screen to start

2.6.6 Screen 6 : Table Select Screen

This screen is displayed if the designer selected a local database, database server or HTML file as the data source. It enables the designer to select a table from which the data are imported.

This screen contains the following controls:

1. *Select Table List*
This is a list of the tables in the data source selected by the designer, of which she can select one.

2. *Next Button*
Pressing this button causes Screen 6 to close, to be replaced by Screen 7

3. *Previous Button*
Pressing this button causes Screen 6 to close, to be replaced by Screen 2.

4. *Cancel Button*
Pressing this button causes the Analyzer to exit

5. *Tutorial Button*
Pressing this button causes the tutorial for this screen to start

2.6.7 Screen 7 : Selecting an analysis template

This screen allows the designer to select a pre-defined analysis template (refer Section 2.6.15 for details on creating an analysis template). In this manner, the designer can apply the same analysis to different tables of the same type (for example, sales figures of different months)

This screen contains the following controls:

1. *Define a new Analysis template Option (default)*
Selecting this option implies that the designer does not wish to use a pre-defined template, but wants to define a new analysis template.

2. *Analyze using a saved template Option*
Selecting this option implies that the designer wishes to use a pre-defined template.

3. *Next Button*
Pressing this button causes Screen 7 to close. If the designer selected the first option (Define a new Analysis template), Screen 9 is displayed. If the designer selected Option 2 (Analyze using a saved template), Screen 8 is displayed.

4. *Previous Button*
Pressing this button causes Screen 7 to close, to be replaced by Screen 2.

5. *Cancel Button*
Pressing this button causes the Analyzer to exit.

6. *Tutorial Button*

Pressing this button causes the tutorial for this screen to start.

2.6.8 Screen 8 : Select Analysis Template Dialog Box

This is the standard system File Select Dialog Box, and allows the designer to select a saved Analysis Template file.

If the designer selects an analysis template file and closes this dialog box, Screen 16 is displayed.

If the designer closes this dialog box without selecting a file, control returns to Screen 7.

2.6.9 Screen 9 : Selecting columns for OLAP queries

This screen allows the designer to select columns that are important for end-user queries. This screen contains the following controls:

1. *Column List*

The columns in the selected table are displayed. The designer can select one or more columns from this table.

2. *Sample OLAP Query Textbox*

This control displays a sample query involving the column(s) selected by the designer. (The sample query displayed changes as and when the column(s) selected by the designer change(s))

3. *Next Button*

Pressing this button causes this screen to close, and Screen 10 to be displayed.

4. *Previous Button*

Pressing this button causes this screen to close, to be replaced by Screen 7.

5. *Cancel Button*

Pressing this button causes the Analyzer to exit.

6. *Tutorial Button*

Pressing this button causes the Tutorial for this screen to start.

2.6.10 Screen 10 : Selecting a Numeric Variable for Data Mining

This screen enables the designer to select the numeric variable for data mining analysis. This screen contains the following controls:

1. *Numeric Variable List*

A list of the numeric variables in the selected table is displayed. The designer can select one numeric variable from this table.

2. Sample Data Mining result Textbox
This textbox displays a sample data mining result involving the numeric variable selected by the designer. (The sample result displayed changes as and when the numeric variable selected by the designer changes)

3. Next Button
Pressing this button causes this screen to close, to be replaced by Screen 11.

4. Previous Button
Pressing this button causes this screen to close, to be replaced by Screen 9.

5. Cancel Button
Pressing this button causes the Analyzer to exit.

6. Tutorial Button
Pressing this button causes the tutorial for this screen to start.

2.6.11 Screen 11 : Selecting Variables of interest for Data Mining

This screen enables the designer to select the variable(s) for data mining. It contains the following controls:

1. Variable List
This is a list of the variables in the selected table. The designer can select one or more variables from this list

2. Sample Data Mining result Textbox
This textbox displays a sample data mining result involving the variable(s) and numeric variable selected by the designer. The sample result displayed should change as and when the designer makes changes in the variable(s) selected for analysis.

3. Next Button
Pressing this button causes this screen to close, to be replaced by Screen 12.

4. Previous Button
Pressing this button causes this screen to close, to be replaced by Screen 10.

5. Cancel Button
Pressing this button causes the Analyzer to exit.

6. Tutorial Button
Pressing this button causes the tutorial for this screen to start

2.6.12 Screen 12 : Selecting a decision variable for Data Mining

This screen enables the designer to select a decision variable for data mining. It contains the following controls:

1. *Decision Variable List*

This list contains potential candidates for the decision variable. The designer can select one variable from this list.

2. *Sample Data Mining result textbox*

This textbox displays a sample data mining result involving the variable(s) selected for data mining, the numeric variable and the decision variable. The sample result displayed should change as and when the designer changes the decision variable selection.

3. *Next Button*

Pressing this button causes this screen to close, to be replaced by Screen 13.

4. *Previous Button*

Pressing this button causes this screen to close, to be replaced by Screen 11.

5. *Cancel Button*

Pressing this button causes the Analyzer to exit.

6. *Tutorial Button*

Pressing this button causes the tutorial for this screen to start.

2.6.13 Screen 13 : Selecting the cutoff for Data Mining

This screen enables the designer to set the cutoff for data analysis. It contains the following controls:

1. *Cutoff textbox*

The designer can enter the desired cutoff in this textbox. The default value is 0.02 (2%)

2. *Next Button*

Pressing this button causes this screen to close, to be replaced by Screen 14.

3. *Previous Button*

Pressing this button causes this screen to close, to be replaced by Screen 12.

4. *Cancel Button*

Pressing this button causes the Analyzer to exit.

5. Tutorial Button
Pressing this button causes the tutorial for this screen to start.

2.6.14 Screen 14 : Analysis Description

This screen enables the designer to enter a title for the data mining analysis she has just programmed (e.g., "Analysis of sales by quarter"). It contains the following controls:

1. Analysis Description Textbox
The designer can enter a description for the data mining analysis in this textbox.

2. Define another data mining question Checkbox
The designer can check this box if she wants to define another data mining question.

3. Next Button
Pressing this button causes this screen to close. If the designer has checked the "Define another data mining question" box, Screen 10 is displayed. Else, Screen 15 is displayed.

4. Previous Button
Pressing this button causes this screen to close, to be replaced by Screen 13.

5. Cancel Button
Pressing this button causes the tutorial for this screen to start.

6. Tutorial Button
Pressing this button causes the tutorial for this screen to start.

2.6.15 Screen 15 : Saving the analysis template

This is the standard system File Save Dialog Box. It enables the designer to save the analysis template for future use, if required.

When this dialog box is closed, Screen 16 is displayed.

2.6.16 Screen 16 : Finish

This screen displays a "Finish" message. It contains the following controls:

1. Finish Button
Pressing this button causes the Analyzer to start processing the raw data. Duing the processing, the Analyzer should display suitable progress messages.

2. Previous Button

Pressing this button causes this screen to close, to be replaced by Screen 14.

3. Cancel Button
Pressing this button causes the Analyzer to exit, without processing the data.

2.7 The Analyzer Online Tutorial

2.8 Issues

2.8.1 Portability

2.8.2 Data Storage

2.8.3 Multiple Users

2.8.4 Internationalization

3 The VirtualMiner Viewer

3.1 User Interface

3.2 Data Mining and OLAP query results

3.2.1 Collation

3.2.2 Printing / faxing

3.2.3 E-mail

3.2.4 Publishing to website

3.3 The Viewer Online Tutorial

4 Typical Usage Scenario

Blazer Engine
OLAP Support
AF Support
Categorical attributes only
Cutoff for size of dataset
N-P Complete problem (cliques)
HTML

Analyzer Component Architecture

Analyzer Component Architecture

1 Introduction

The Analyzer is a software component that sets up and prepares data for analysis using OLAP as well as data mining techniques. It can be thought of as a software "IC" that can be plugged into an existing program by a developer. The architecture of the Analyzer component is shown in Figure 1. As can be seen, the Analyzer does not have a user interface of its own. Thus, it can be plugged into the user interface of an existing program, or a customized user interface can be designed exclusively for it. Either way, it provides a standard set of interfaces so that its functionality can be made use of.

2 Analyzer Modules

The Analyzer component has 8 main modules. These are described in brief below:

2.1 Input Processor Module

The Input Processor Module is responsible for processing user inputs (which could come from a Graphical User Interface, or a Command Line Interface, useful for batch processing), and passing them on to the Data Handler Module. These inputs are of the following types:

2.1.1 Raw Data Inputs

These inputs relate to the data to be processed – the kind of data (ASCII Text, HTML, Spreadsheet, local database, remote database, real-time data), the name of the data table, etc.

2.1.2 OLAP Inputs

The user can specify the columns of the data to be included for OLAP analysis.

2.1.3 Attribute Focusing Inputs

The user can specify various parameters for data mining using Attribute Focusing – the Focus Attributes, the Numeric Attribute, the Decision Variable and the cutoff

2.1.4 Clique Inputs

The user can specify whether or not clique analysis is to be performed on the data.

2.1.5 Grammar Inputs

The user can specify the grammar associated with various columns in the data.

2.1.6 Control Inputs

The user can provide various control inputs, such as “Start Data Import”, “Stop Data Import”, “Start Analysis”, “Stop Analysis”, etc.

2.1.7 Queries

The user can query the status of various operations, such as “Is Data Import Over”, or “Is Data Analysis Over”, or “Get Status”, or “Get Last Error”, or “Get Data Mining Sample Result”. (These are provided so that the User Interface designer can provide appropriate feedback to the user)

2.2 Data Handler Module

The Data Handler Module acts as the “CPU” of the Analyzer. It controls the imports and analysis of several sets of data. It performs the following tasks:

- Sets up internal data structures according to the inputs provided by the Input Processor Module for each dataset
- Co-ordinates Data Import of a dataset with the Data Access Module, and stores the raw data imported by the Data Access Module
- Co-ordinates analysis of a dataset with the Blazer Module

2.3 Data Access Module

This module is responsible for importing different kinds of data into the Analyzer, and passing the data into a message queue for analysis by the Blazer Module. The import functionality of the Blazer functions within a separate thread of control. This is to enable the user to cancel the import operation, if required.

2.4 Blazer Module

The Blazer Module is responsible for data analysis. It follows a proprietary data mining algorithm. The analysis functionality also runs within a separate thread of control, so that the user can cancel the operation, if required.

2.5 Output Processor Module

The Output Processor Module is responsible for storing the analysis results in a suitable format on the hard disk. It also stores the grammar associated with the raw data. It is controlled by the Blazer Module, and thus operates within the same thread as the analysis function of the Blazer.

2.6 Example Generator Module

The Example Generator Module generates sample data mining results, based on the inputs provided to it (the attributes involved, the decision variable, etc.). Thus, the user can get a “quick look” at the kind of results produced based on the inputs she has provided.

2.7 Error Handler Module

The Error Handler Module handles all errors encountered by the other modules. It also provides a textual description of the last error encountered. It is responsible for graceful exit, in case of a fatal error.

2.8 Status Handler Module

The Status Handler Module is responsible for providing status reports to the user. The other modules constantly feed this module with the latest status of their operations.

Figure 1: Analyzer Component Architecture

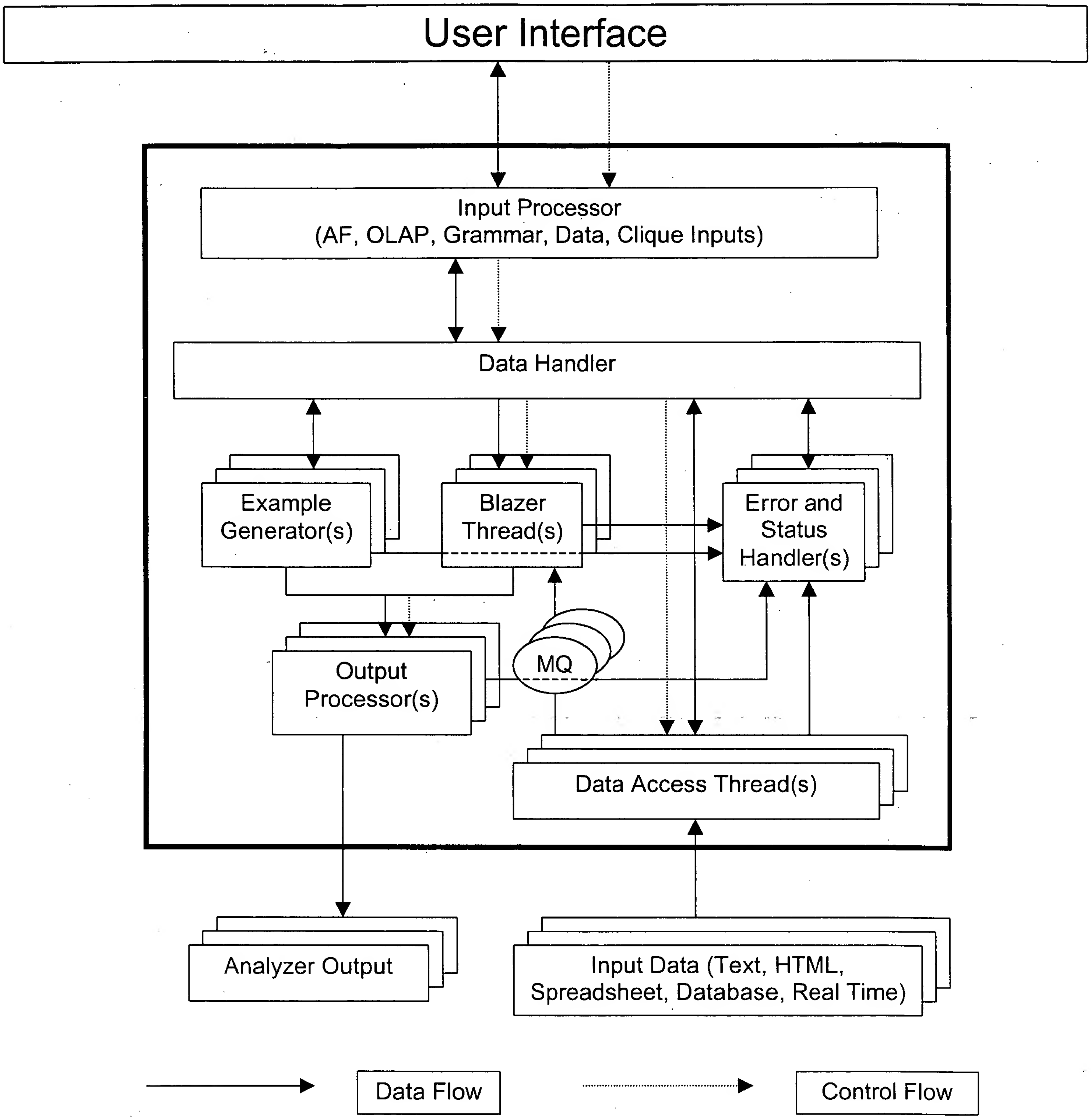
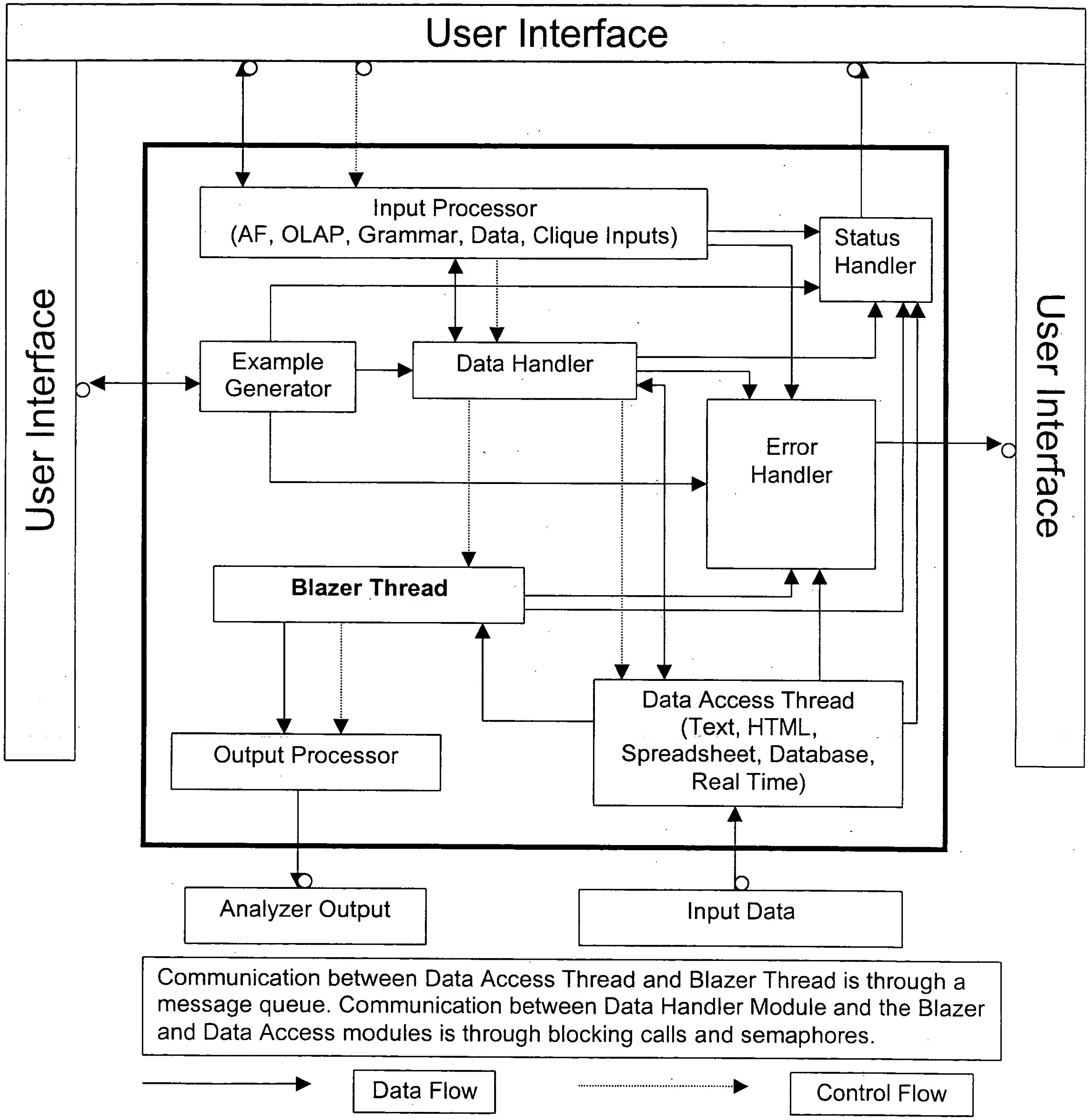


Figure 1: Analyzer Component Architecture



Communication between Data Access Thread and Blazer Thread is through a message queue. Communication between Data Handler Module and the Blazer and Data Access modules is through blocking calls and semaphores.

Viewer Component Architecture

Viewer Component Architecture

1 Introduction

The Viewer is a software component that enables the user to analyze data (pre-processed using the Analyzer component) using OLAP as well as data mining techniques. It can be thought of as a software "IC" that can be plugged into an existing program by a developer. The architecture of the Viewer component is shown in Figure 1. As can be seen, the Viewer does not have a user interface of its own. Thus, it can be plugged into the user interface of an existing program, or a customized user interface can be designed exclusively for it. Either way, it provides a standard set of interfaces so that its functionality can be made use of.

2 Viewer Modules

The Viewer has 6 main modules. These are described in brief below:

2.1 Query Processor Module

The Query Processor Module is responsible for processing user queries. These could be of the following types:

- Data Mining Queries
  For example, "show me all patterns involving green shirts"
- OLAP Queries
  For example, "What was the revenue from green shirts last winter ?"
- Clique Queries
- Status Queries
  For example, "Get Status", or "Get Last Error"

The Query Processor Module passes these queries on to the Query Handler Module.

2.2 Data Processor Module

The Data Processor Module is responsible for importing the pre-processed data output by the Analyzer Component.

2.3 Query Handler Module

The Query Handler Module acts as the "CPU" of the Viewer Component. It performs the following tasks:

- Sets up internal data structures according to the inputs provided by the Query Processor Module
- Initiates data import by the Data Processor Module
- Runs queries on the data in a separate thread of execution
- Passes results of queries on to the Result Generator Module

- Returns result text obtained from the Result Generator Module to the user

2.4 Result Generator Module

The Result Generator Module is responsible for forming textual descriptions of query results. It formulates these descriptions based on the query results obtained from the Query Handler Module, and the Grammar description obtained from the Data Processor Module.

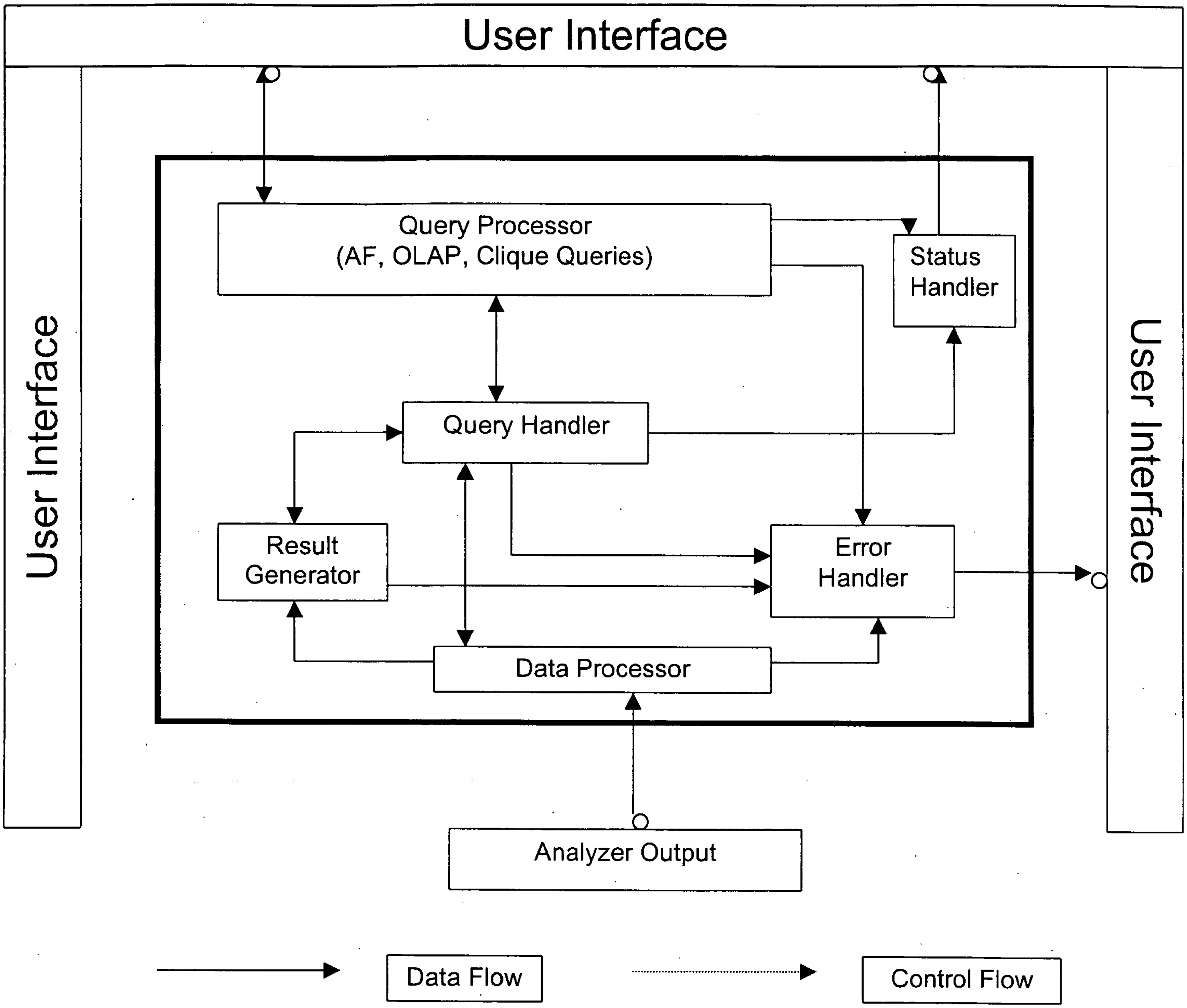
2.5 Error Handler Module

The Error Handler Module handles all errors encountered by the other modules. It also provides a textual description of the last error encountered. It is responsible for graceful exit, in case of a fatal error.

2.6 Status Handler Module

The Status Handler Module is responsible for providing status reports to the user. The other modules constantly feed this module with the latest status of their operations.

Figure 1: Viewer Compon nt Architecture



ActiveTables™ for mining data on the Internet

A wide variety of tabular data exists on the World Wide Web. This ranges from sports scorecards (NBA, NFL, etc.) to annual financial reports of companies. This document discusses, in brief, a mechanism for mining these data and providing additional interesting information about the data to the viewer, within the confines of a Java-enabled web browser.

Problem Definition

A mechanism has to be developed whereby additional information can be obtained about the field(s) of a table that the user is viewing, and a way in which this information can be mined along with additional sources of information, to provide interesting patterns to the user. Ideally, this information should appear in a floating balloon, which constantly updates itself, as the user moves his/her mouse around in the table.

A table is represented in a web page as a series of HTML tags. Consequently, there is no way in which an application (other than the browser) can tell which column / row of the table the user's mouse is currently at. Also, there is no means, currently, of defining additional sources of information.

The ActiveTable™ mechanism is proposed to replace the current "passive" representation of tables through HTML tags.

What is an ActiveTable™ ?

An ActiveTable™ is an applet that contains the table to be mined. It is embedded in an HTML page. It is itself a JavaBean, and consists of the following JavaBeans :

(1) A set of **GridElements**. Each GridElement represents a single cell in the table. Each grid element "knows" its position (row/column) in the table, as well as the datum that it contains. Each GridElement is capable of responding to mouse movements over its visual representation in the ActiveTable™

(2) One or more **buttons**, e.g., to turn on/off data mining, to ask user defined questions, etc.

(3) A Floating **Balloon**, which would display interesting results from the current table. This updates continously, depending on the position of the user's mouse.

(4) A **text area**, which would display interesting data mining results and answers to user questions by combining the contents of the current table with other data from the web.

(5) Some other elements of pure "**decorative**" value, such as horizontal and vertical lines, bitmaps, etc.

Creating an ActiveTable™

An ActiveTable™ is a JavaBean, and can be inserted into a web page (?), or any other Java application, just like any other JavaBean, using a visual builder tool (such as IBM VisualAge, or Symantec Visual Café).

When a developer instantiates an ActiveTable™, a set of wizards appear, enabling him/her to customize the ActiveTable™. The wizards would require answers to questions such as the following :

(1) What is the **primary data source** of the ActiveTable™ ? (the primary data source of an ActiveTable™ refers to the source for the data actually displayed to the user by the ActiveTable, in its GridElements. This could be a subset of or an entire text file, ODBC table, etc). This information is **required**.

(2) What are the **secondary data sources**, if any ? (Secondary data sources refer to additional data sources that can provide more information to be mined. These could be text files, ODBC tables, text reports, video clips, etc.) This information is **optional**.

Once the primary data source for an ActiveTable™ has been identified, an applet representing it will be built by the builder tool, and inserted into the specified HTML page.

How will the ActiveTable™ work ?

When the user sees the HTML page, the ActiveTable™ applet will be invoked automatically by the browser. When she moves her mouse around on the table, the particular GridElement on which the mouse is located, will respond. The GridElement will perform some standard calculations, and update the floating balloon. For example, if the user's mouse is on the FGM column for Michael Jordan in the following table (row = 2, column = 2), the appropriate GridElement would respond.

Player	FGM	FGA
Jordan	5	10
Pippen	7	10
Kerr	6	8

Since the GridElement "knows" that it belongs to the 2nd row and 2nd column, it could display a message such as the following :

"Did you know that Michael Jordan made 5 out of the Bulls' 18 Field Goals (28 %) ?"

Alternatively, the GridElement could act more intelligently. Since it "knows" it belongs to the "FGM" column, it could query the "FGA" column, and come up with :

"Did you know that Michael Jordan shot 5 for 10 (50 %), whereas the Bulls shot 18 for 28 (64 %) ?"

Note that the first example could **always** work, in any domain, whereas the second represents a customization for a specific (in this case, basketball) domain.

When the user clicks on the Data Mining button, the ActiveTable™ would look up data from the secondary sources specified by the developer, and mine the results together. (The developer would have to provide some additional information while customizing the ActiveTable™, such as the Focus Attributes, etc.)

The user could click on the Ask Questions button, to ask her own questions. These questions would also be fired across all the (primary and secondary) data sources associated with the ActiveTable™

A typical DM / user question answer would be displayed in the answer text area, and would look like this : "Did you know that in this season, against the Knicks, in the games in which Jordan had more than 10 rebounds, the Bulls won 3 of 4 games (75%) "

Other Issues :

(1) The ActiveTable™ JavaBean can be converted into an ActiveX control, thus enabling the same functionality from within MS-Office containers, such as MS-Word, MS-Excel, etc.

(2) The ActiveTable™ could first be made available in its entirety to developers. The developer would be able to customize an ActiveTable™, by specifying different data sources, etc. Later, the various parts of the ActiveTable™ (buttons, text areas, etc.) could be made available individually, so that developers can customize the look and feel of the entire ActiveTable™ to their liking.

(3) The user can specify a text file / ODBC data sources as the secondary data source. In order to mine data, the current ActiveTable™ has to query / import data from these sources. This has various disadvantages, e.g., it assumes the existence of a database server (in case of ODBC), and it also involves querying, and sometimes, parsing files for importing data. However, if these secondary data sources are themselves represented using ActiveTables™ (i.e., they are the primary data sources of other ActiveTables™), then, using the RMI (Remote Method Invocation) mechanism available in Java, it might be possible to query these ActiveTables™ for secondary data, thus ensuring truly distributed computing. For example, consider the following scenario :

In the above example, the user clicks on data mining, while viewing Jordan's Field Goal Percentage. The ActiveTable™ issues a query to the NBA server, asking something like

"Any ActiveTables™ out there, who have data about the Bulls against the Knicks, please respond with some data (related to Shooting) for the game."

This query goes out to all ActiveTables™ on the NBA server, and evokes reponses from all the ActiveTables™ with primary data related to Bulls-Knicks games. The client ActiveTable™ can then use these data to perform some analysis, and show the results to the user.

Some comments on this mechanism :

a. It is totally independent of the way the data are stored on the server (text files, ODBC, etc.)
b. It does not overload any database server (however, it does cause additional burden to the HTTP server). From the point of view of security, a large company would prefer requests to the HTTP server rather than allowing millions of users direct access to their databases.
c. It involves minimal server-side programming (just setting up the mechanism by which ActiveTables™ could respond to these remote requests)
d. Data within the ActiveTable™ could be intelligently indexed for faster retrieval
e. Information obtained from various ActiveTables™ could be cached by the client ActiveTable™ for re-use
f. However, it does not make use of the database server's inherent optimization with respect to SQL queries.

Unclear at this point :

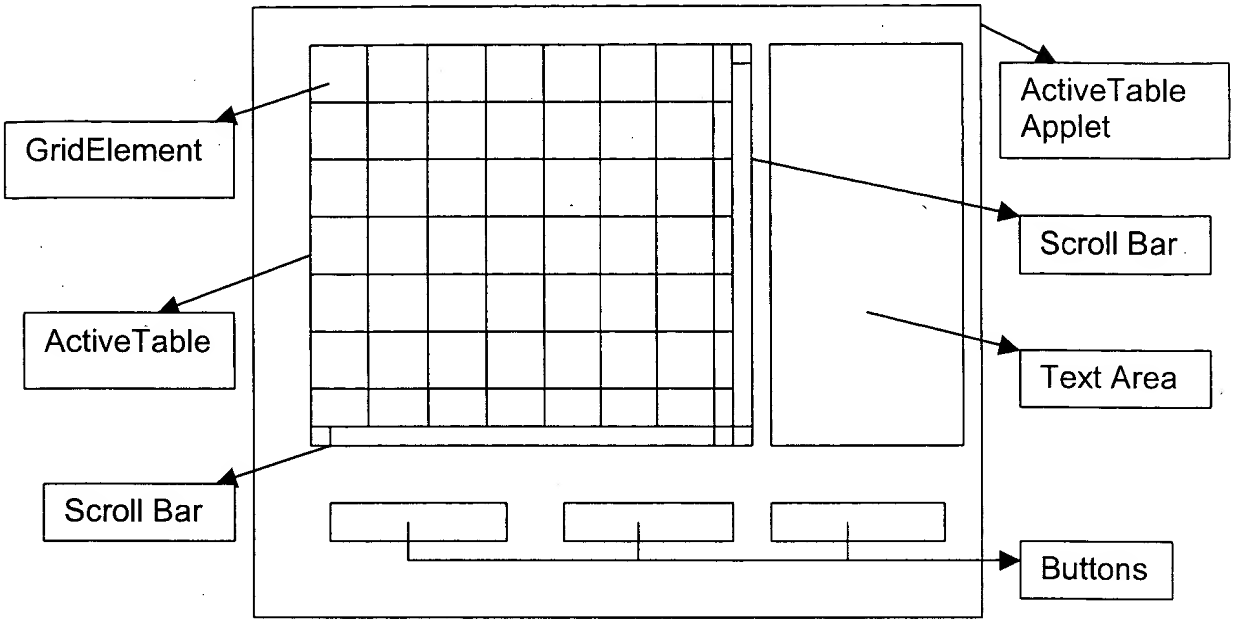
a. Once the ActiveTable™ has been customized by the developer, it can be saved to a ".ser" file. This will ensure that it retains all the properties that the developer has set. The ".ser" file can then be opened by a browser, like an applet, through an extension to HTML. However, as of now, no browser supports this serialized version of Java class files.
b. RMI requires more study to determine that all that has been mentioned above, is, in reality, possible to implement

Phase 1 of the project:

- Develop the beans for implementation of ActiveTable™.
- Implement the *customizer* for the ActiveTable™, where the ActiveTables™ is based on a single data source. (No filter for the data)
- Make the grids of the ActiveTable™ responds to mouse movements and pop up with simple but interesting statistical calculations on the data from the primary data source of the ActiveTable™.
- Start working on the re-design of the data mining algorithm, and the presentation of data mining results.

ActiveTable™ Design

The ActiveTable™ will appear as shown below :



ActiveTable Viewer Software Requirement Specifications

R. Krishna Kumar

Virtual Gold Confidential

1 Introduction

System Description
Platforms
Invocation through browser

2 Analysis Details

Units of Analysis : Decision Variable, Focus Attributes, Metric – pre-decided
Data processed and stored

3 ActiveTable Functionality

1. Invoked from server
2. Checks for license availability
3. Downloads raw data
4. Displays ActiveTable to user with raw data
5. Starts download of processed data
6. Answers user questions
7. Displays answers and interesting patterns in output area
8. Displays tutorial, if required, in output area

4 ActiveTable User Interface

The ActiveTable Screen consists of two regions : the ActiveTable area and the output / tutorial area, as shown in Fig. 4-1.

ActiveTable Area
Output Area : Result Mode and Tutorial Mode

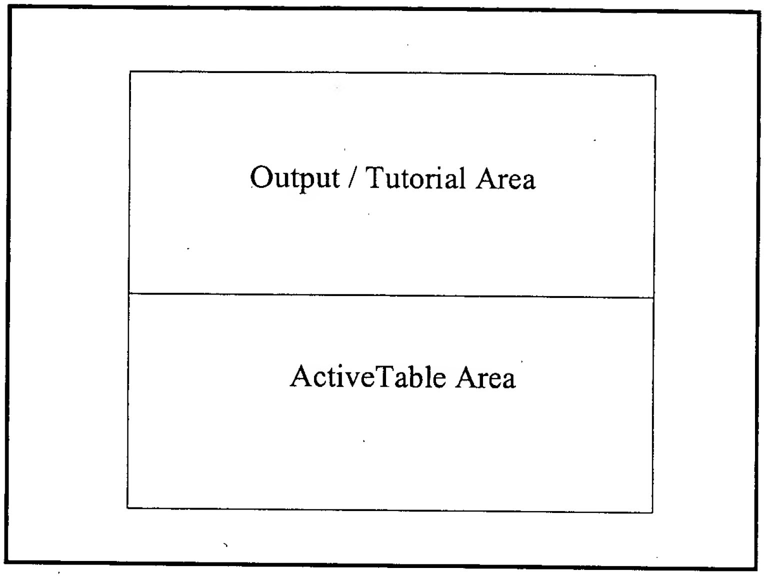
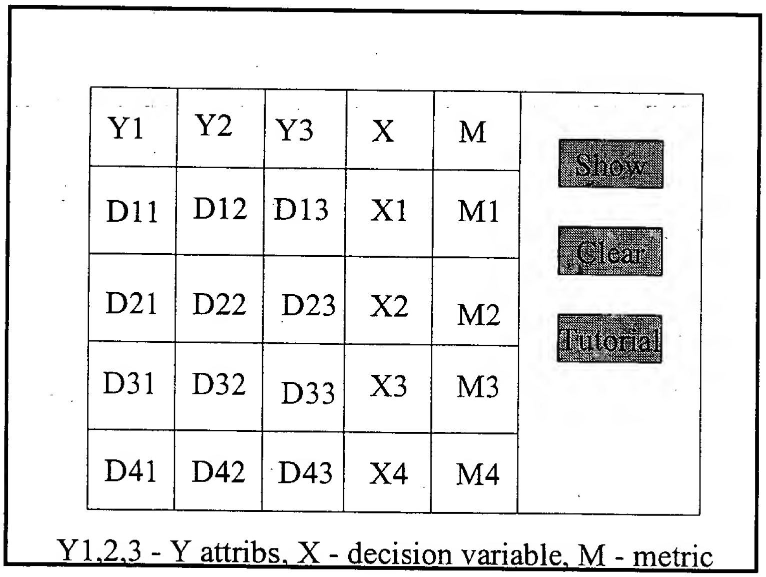


Figure 4-1: ActiveTable Viewer User Interface

The details of the ActiveTable area are shown below in Fig. 4-2:

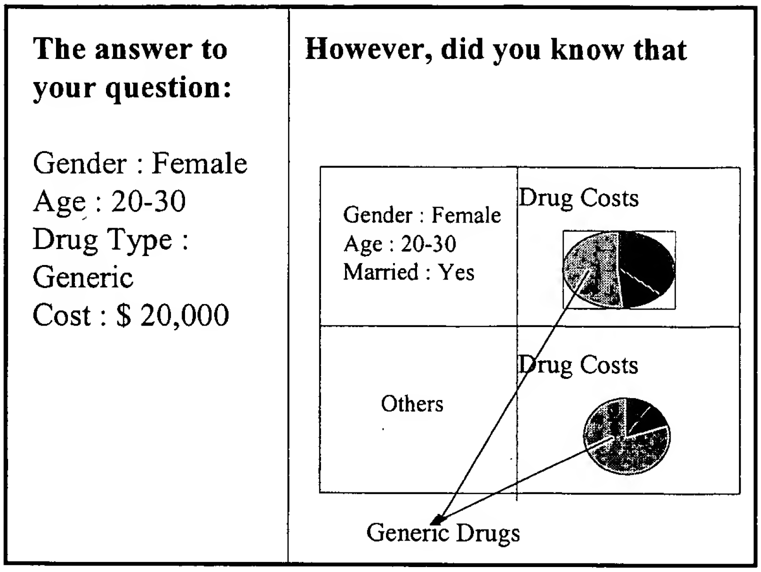


Y1	Y2	Y3	X	M
D11	D12	D13	X1	M1
D21	D22	D23	X2	M2
D31	D32	D33	X3	M3
D41	D42	D43	X4	M4

Y1,2,3 - Y attribs, X - decision variable, M - metric

Figure 4-2: ActiveTable area details

The ourput area can display the results of user questions (“Result Mode”), or a tutorial (“Tutorial Mode”). An example of the output area in results mode is shown in Fig. 4-3.



The answer to your question:

Gender : Female
Age : 20-30
Drug Type : Generic
Cost : $ 20,000

However, did you know that

Figure 4-3: Output Area in Result Mode

ActiveTable Design

1 Introduction

This document details the design of ActiveTable, a tool used for OLAP and OM (Online Mining) queries. ActiveTable is an applet, and is invoked through a web browser. It works off data that have been pre-processed by another tool codenamed Blazer.
Blazer output data are loaded from a server by ActiveTable, and then displayed to the user for OLAP and OM queries.

An object-oriented approach has been taken in the ActiveTable design. The main classes are described in the following sections.

1.1 Document Conventions

The following conventions have been followed in this document:

- Class types are described in **bold**
- Objects (class instances) are described in ***bold Italics***
- Member variables are described in *small italics*
- Member constants are described in *CAPITAL ITALICS*
- Other variables are in *`courier italics`*
- Member functions are described in `courier`

1.2 Definitions

OLAP	On Line Analytical Processing
OM	Online Mining
Blazer	A tool used for pre-processing data, rendering it suitable for OLAP / OM

2 Blazer and ActiveTable

As mentioned in Section 1, ActiveTable works off data pre-processed by Blazer. These data are stored on files on the web server. The ActiveTable applet is invoked with these file names as parameters. It then loads the data from these files, enabling the user to then run OLAP and OM queries.

2.1 Blazer Analysis

Blazer is used by developers / IS managers to run an analysis on input data. An analysis consists of

- a set of focus attributes,
- a decision variable and
- a metric.

Once these parameters of the analysis have been specified, Blazer processes the input data, and outputs the results into files, stored on the web server. Thus, each Blazer output consists of the results of a single analysis on one input dataset.

2.2 Blazer Results

The results output by Blazer are of three types:

2.2.1 "Raw" data

"Raw" data refers to the rows of the original data input to Blazer. These are stored by Blazer to enable the user of ActiveTable to view the original data. To save storage space, Blazer stores only unique rows of the input data, after consolidating the metric information. Thus, for example, for the input data shown in Table 1, Blazer would store the raw data as shown in Table 2.

Item	Color	City	Revenue
Shirt	Green	Manhattan	$15
Tie	Blue	Chicago	$10
Shirt	Green	Manhattan	$25
Tie	Blue	Chicago	$5

Table 1: Sample input data

Item	Color	City	Revenue
Shirt	Green	Manhattan	$40
Tie	Blue	Chicago	$15

Table 2: "Raw" data stored by Blazer for data in Table 1

Consequently, Blazer requires one file to store "raw" data.

2.2.2 "Processed" data

"Processed" data refers to the results output by Blazer after mining the input data. Blazer output actually consists of an array of n-tuples (where n is the number of focus attributes + 1), along with a metric associated with each n-tuple. Each attribute of the Blazer analysis has a designated placeholder in the n-tuple, for storing the value of the attribute. An n-tuple is thus a concatenation of attribute values.

Not all attributes may have values in the n-tuple. if an attribute does not have a value in the n-tuple, the placeholder for the attribute will be set to "NULL".

To reduce memory requirements, the attribute values are mapped to (non-zero) byte Ids. Each n-tuple is thus represented as a vector with n bytes, each byte representing the ID of the value of the associated attribute. A "NULL" value is represented by an ID of 0.

Consequently, typical Blazer output looks like this (assuming an analysis with 4 focus attributes, or n = 5):

5-tuple	Metric
1 0 23 10 1	2.3
1 1 20 12 1	1.2
2 1 31 12 1	0.7
……	…
……	…
……	…

Note that the array is ordered by the n-tuples. Each n-tuple is treated as a number, with the first element of the n-tuple as the most significant "digit", and the last element as the least significant "digit".

The array can now be searched to find any n-tuple. However, to reduce the search time, a further optimization can be done. Each attribute value stores the row numbers of the array in which it appears. Thus, the n-tuples in the first 3 rows of the above table can be represented as:

Attribute 1 – Value ID 1 – Rows 1, 2
Attribute 1 – Value ID 2 – Row 3
Attribute 2 – Value ID 1 – Rows 2, 3
Attribute 3 – Value ID 20 – Row 2
Attribute 3 – Value ID 23 – Row 1
Attribute 3 – Value ID 31 – Row 3
Attribute 4 – Value ID 10 – Row 1
Attribute 4 – Value ID 12 – Rows 2, 3
Attribute 5 – Value ID 1 – Rows 1, 2, 3

Once the n-tuples have been represented in this manner, only the metrics need be stored in the array:

Metric
2.3
1.2
0.7
…
…
…

Thus, the row number for an n-tuple can be determined by finding the intersection of the row numbers of its constituent values. Once the row number has been found, the etric for the n-tuple can be read from the metric array.

Consequently, Blazer requires 2 files to store "processed" data – one for the attribute-value Ids and row numbers, and the other for the metric array.

2.2.3 Interesting Patterns

The data mining patterns found by Blazer are also stored. A typical data mining pattern highlights the difference in the proportion of a particular value of the decision variable for a subset of the population (the "interesting pattern") as opposed to the proportion in the entire population (the "base" pattern). For example, "the bulls scored in 23% of their possessions when Jordan was 2Guard ("interesting"), but scored in 56% of their overall possessions ("base")".

It is clear that both the base and the interesting patterns need to be stored to represent a data mining pattern in its entirety. However, the base pattern (i.e., the values of the metric for different values of the decision variable) is the same for all interesting patterns, and thus needs to be stored only once.

Consequently, the Blazer output for the interesting patterns consists of the following information:

- Base pattern (array of metrics for different values of the decision variable)
- Number of interesting patterns
- For each interesting pattern,
  - Vector indicating the pattern (n-tuple of value Ids)
  - Array of metrics for different values of the decision variable. There are 2 possible storage mechanisms here:
    - Store the metrics for all values of the decision variable for the interesting pattern (in the above example, this would mean storing metric values for "scored", "missed" and "turned over" for the possessions when Jordan was 2Guard)
    - Store only the metric for the interesting value of the decision variable, and bunch metrics for all other values of the decision variable under "others" (in the above example, this would mean storing the metric values for "scored", and bunching those for "missed" and "turned over" under "others". This mechanism could be useful when the decision variable has a large number of values

Consequently, Blazer requires one file to store pattern information.

3 Invoking ActiveTable

ActiveTable enables the user to view data and ask OLAP /OM questions of the data, and view output reports. When ActiveTable is invoked, it should be "aware" of the name of file that contains analysis information, encapsulated within an **CVGIAnalysis** object. To achieve this, it could be invoked with this filename as a parameter.

Note 1: An alternative mechanism would be to serialize this information during the creation of ActiveTable, so that it is automatically re-loaded when ActiveTable is invoked. The latter mechanism is preferable.

The **CAnalysis** object, in turn, contains the names of the files containing data pre-processed by Blazer. These can then be loaded from the appropriate files. This 2-tier scheme of invocation is useful for separating the various loading processes, as explained below.

The input to ActiveTable consists of data pre-processed by using Blazer. However, the "processed" data may take considerable time to load onto the client computer. Hence, the "raw" data are loaded first, so that the user can at least view the data. The "processed" data are loaded in a separate background thread, to cause minimal discomfort to the user.

Four data structures are required to enable this separation of "raw" and "processed" data:

1. **CVGIGrid** – contains unique rows of "raw" data, in the form of n-tuples of attribute and value names and Ids. This information is loaded first from the web server. (refer Section 2.2.1 for more information)
2. **CVGIAttributeSet** – contains the names of the attributes (focus attributes, decision variable and metric), their unique values (for the focus attributes and decision variable), Ids for these values, and indices to the processed data structure. (refer Section 2.2.2 for more information)
3. **CVGIMetricSet** – an array containing the metrics of all n-tuples in the "processed" data. The Index information stored in **CVGIAttributeSet** points to rows of this array (refer Section 2.2.2 for more information)
4. **CVGIPatternSet** – information about interesting data mining patterns for the data. (refer Section 2.2.3 for more information)

The steps followed by the viewer upon invocation are listed below:

1. User invokes Viewer applet, using a web browser
2. **CVGIActiveTable** object created on user's computer. The *analysisInformation* object is de-serialized from the serialized file, or loaded from the file name specified as a parameter.
3. **CVGIActiveTable** object checks for required permission with the **CVGILicenseServer** object on the server (using remote object methods). This step may be omitted for prototyping purposes.
4. "Raw" data are loaded from the server:
    - **CVGIActiveTable** loads **CVGIGrid** from the server. This contains n-tuples of unique rows of "raw" data

5. The “processed” data are loaded from the server:
    - **CVGIActiveTable** loads **CVGIAttributeSet** from the server. This contains attribute, value Ids and indices to the metric array
    - The Metric information is loaded from the server, through the **CVGIMetricSet** object
    - The Pattern Information is loaded from the server, using the **CVGIPatternSet** objects
6. The user can now ask OLAP/OM questions, and results are displayed

1. Introduction

ActiveTable™ is a product for enabling data mining and querying on tabular data present in web pages. It is a customizable component that can be easily manipulated using a builder tool, enabling developers to quickly create a customized data querying and mining application for their web pages.

ActiveTable consists of three sub-components :

1. The **FastPattern** component, which enables shows a user some quick context-sensitive results based on elementary data mining on a single data source.
2. The **UserQuery** component, which enables users to ask their own questions, using multiple data sources
3. The **DataMining** component, which enables users to ask complex data mining questions, across multiple data sources.

In accordance with the sub-components mentioned above, ActiveTable™ development will proceed in three stages, viz. :

1. Development of basic ActiveTable components, and all classes required to support the **FastPattern** mechanism
2. Support for the **UserQuery** component, and multiple data sources
3. Support for the **DataMining** component

This document addresses the design for the first stage of ActiveTable development, viz. Basic ActiveTable components, and **FastPattern** support.

Section 2 deals with the ActiveTable architecture, and discusses the component classes in detail.

1.1. Document Conventions

Throughout this document, Class and Interface names will be in **bold**. Methods and properties of a class will be in ***bold italics***. Other variables will be in *italics*.

2. Components of ActiveTable™

ActiveTable will be implemented as a JavaBean, that can be visually manipulated and customized using a builder tool.

2.1. ActiveTable : Basic Functionality

ActiveTable requires some basic functionality to be put in place, before further additions (such as the FastPattern mechanism) can be addressed. This includes methods for creating and customizing the ActiveTable. This section deals with all the classes required for supporting this basic functionality.

2.1.1. The **ActiveTable** *Class*

The **ActiveTable** class is the basic class required for implementing an ActiveTable. This class extends the **Applet** class, and is a JavaBean. In the first stage of the ActiveTable implementation, this is the only JavaBean that is exposed to the developer.

The **ActiveTable** class contains all information related to the data in the table (including indexing mechanisms for fast data retrieval), as well as information about the look and feel of the data on the user's screen.

In addition, for FastPattern implementation, the ActiveTable class contains information related to the analyses to be performed for the user.

An ActiveTable must have a Primary Data Source. This is the source of the data displayed on the user's screen. In addition, it could have one or more Secondary Data Sources. These are the sources for data that are pulled in when the user mines data across different data sources. Only Primary Data Sources are supported in this version of ActiveTable. Data from the Primary Data Source are imported into the ActiveTable when it is created by the developer. An indexing mechanism (using the **Index** class) is then created for this table.

An ActiveTable also contains a two-dimensional array of **GridElements**. These represent the cells of the table displayed by the ActiveTable.

2.1.1.1 Constructors

2.1.1.1.1 Public ***ActiveTable*** ()

This constructor creates a new **ActiveTable** object with all properties set to NULL.

2.1.1.1.2 Public **ActiveTable** (**DataSourceTypeEnum** *newDataSourceType*, String *newPrimaryDataSource*)

This constructor creates a new **ActiveTable** object with the ***primaryDataSourceType*** property set to newDataSourceType, and the **primaryDataSource** property set to *newPrimaryDataSource*. It also does the following :

1. Creates a new instance of a **DataSource** class by importing data from *newPrimaryDataSource*.
2. Creates ***tabularData [ ]*** (an array of objects of class **GridElement**), using the **DataSource** object created above, and default values for grid size, font weight, font size and color.
3. Destroys the **DataSource** object created above.

2.1.1.1.3 Public ***ActiveTable*** (**DataSourceTypeEnum** *newDataSourceType*, String *newPrimaryDataSource*, **AttributeCluster** ***newFastPatternAnalyses[ ]***)

This constructor creates a new **ActiveTable** object with the ***primaryDataSourceType*** property set to newDataSourceType, the **primaryDataSource** property set to *newPrimaryDataSource*, and the ***fastPatternAnalyses[ ]*** property set to *newFastPatternAnalyses[ ]*. It also does the following :

1. Creates a new instance of a **DataSource** class by importing data from *newPrimaryDataSource*.
2. Creates ***tabularData [ ]*** (an array of objects of class **GridElement**), using the **DataSource** object created above, and default values for grid size, font weight, font size and color.
3. Creates ***tableIndex***, an index of the values appearing in each of the **AttributeCluster**s in the ***fastPatternAnalyses*** array, and ***fastPatternTotals***, an array containing totals of the columns in the ***fastPatternAnalyses*** array (this is concurrent with Step 2, to avoid multiple passes on the data)
4. Destroys the **DataSource** object created above.

2.1.1.2 Properties

2.1.1.2.1 Private **DataSourceTypeEnum** ***primaryDataSourceType***

PrimaryDataSourceType contains the Primary Data Source Type (ASCII / ODBC / HTML)

2.1.1.2.2 Private String ***primaryDataSource***

primaryDataSource is a string that contains the name of the Primary Data Source of the ActiveTable.

2.1.1.2.3 Private **GridElement** ***tabularData[ ]***

TabularData is a two-dimensional array of **GridElements**, representing the actual table displayed.

2.1.1.2.4 Private Dimension ***defaultGridSize***

defaultGridSize contains the default grid size for the grid elements.

2.1.1.2.5 Private int ***defaultFontWeight***

DefaultFontWeight contains the default font weight of the text displayed in the grid element.

2.1.1.2.6 Private int ***defaultFontSize***

DefaultFontSize contains the default font size for the text displayed in the grid element.

2.1.1.2.7 Private Color ***defaultFontColor***

DefaultFontColor contains the default font color of the text displayed in the grid element.

2.1.1.2.8 Private **Index *tableIndex***

TableIndex contains indexing information about the table displayed by the ActiveTable.

2.1.1.2.9 Private **AttributeCluster *fastPatternAnalyses[ ]***

FastPatternAnalyses is an array of **AttributeCluster**s that facilitate FastPattern analysis.

2.1.1.2.10 Private **ColumnInfo *fastPatternTotals [ ]***

FastPatternTotals is an array containing information about the totals for various columns appearing in the ***fastPatternAnalyses*** array.

2.1.1.2.11 Private **InfoBalloon *fastPatternBalloon***

FastPatternBalloon contains the FastPattern information to be displayed to the user when she moves her mouse around on the ActiveTable.

2.1.1.2.12 Private **FIFO *fastPatternCache***

FastPatternCache contains a cache of the previous FastPatterns viewed by the user.

2.1.1.3 Methods

2.1.1.3.1 Public void ***setPrimaryDataSourceType*** (**DataSourceTypeEnum** *newPrimaryDataSourceType*)

Sets ***primaryDataSourceType*** to *newPrimaryDataSourceType*.

2.1.1.3.2 Public **DataSourceTypeEnum *getPrimaryDataSourceType*** ()

Returns ***dataSourceType***

2.1.1.3.3 Public void ***setPrimaryDataSource*** (String *newPrimaryDataSource*)

Sets ***primaryDataSource*** to *newPrimaryDataSource*.

2.1.1.3.4 Public String ***getPrimaryDataSource*** ()

Returns ***primaryDataSource***.

2.1.1.3.5

2.1.2. The ActiveTableBeanInfo Class
2.1.3. The GridElement Class
2.1.4. The DataImport Class
2.1.5. The AsciiDataImport Class
2.1.6. The ODBCDataImport Class
2.1.7. The HTMLDataImport Class
2.1.8. The Index Class
2.1.9. The GridElement Class

2.2. FastPattern Support
2.2.1. The Analysis Class
2.2.2. The Balloon Class
2.2.3. The FIFO Class

The various elements of the ActiveTable Applet are explained below :

1. ActiveTable Applet
2. GridElement
3. ActiveTable
4. Text Area
5. Buttons

2.3. GridElement (extends TextArea)

A GridElement is an object that contains a single element of the data to be displayed and analyzed. It is not a JavaBean, and is not directly available for manipulation by the developer, but serves as a building block for an ActiveTable.

2.3.1. Constructors

2.3.1.1 Public Void GridElement()

Constructs a "floating" GridElement object of default size with data = 0.0.

2.3.1.2 Public Void GridElement(int row, int col, float data)

Constructs a GridElement object of default size anchored to the specified row and column, with data as specified.

2.3.1.3 Public Void GridElement (int row, int col, Dimension Size)

Constructs a GridElement object of the specified size, anchored to the specified row and column.

2.3.1.4 Public Void GridElement (int row, int col, Dimension Size, float data)

Constructs a GridElement object of the specified size, anchored to the specified row and column, and containing the data specified.

2.3.2. Properties

2.3.2.1 Private int row

Specifies the row of the ActiveTable to which the GridElement is anchored.

2.3.2.2 Private int column

Specifies the column of the ActiveTable to which the GridElement is anchored.

2.3.2.3 Private Dimension size

Specifies the size of the GridElement.

2.3.2.4 Private boolean numeric

Specifies whether the GridElement contains numeric data. If this element is true, the GridElement contains numeric data. If it is false, it contains string (textual) data.

2.3.2.5 Private float data

Specifies the data contained in the GridElement. (if isNumeric is true)

2.3.2.6 Private String displayText

Specifies the text displayed in the GridElement when it appears in an ActiveTable. DisplayText always contains the string representation of the data contained in the GridElement.

2.3.2.7 Private int fontSize

Specifies the font size for the GridElement

2.3.2.8 Private int fontWeight

Specifies the font weight for the GridElement

2.3.2.9 Private boolean bold

Specifies whether the GridElement contents are to be displayed in **bold** font.

2.3.2.10 Private boolean italicized

Specifies whether the GridElement contents are to be displayed in *italics*.

2.3.2.11 Private Color color

Specifies the color of the GridElement contents

2.3.3. Methods

2.3.3.1 Public int getRow ()

Returns the row (value of the *row* property) to which the GridElement is anchored.

2.3.3.2 Public void setRow (int newRow)

Sets the row to which the GridElement is anchored (sets the *row* property to newRow)

2.3.3.3 Public int getColumn ()

Returns the column (value of the *column* property) to which the GridElement is anchored.

2.3.3.4 Public void setColumn (int newColumn)

Sets the column to which the GridElement is anchored (sets the *column* property to newColumn)

2.3.3.5 Public Dimension getSize ()

Returns the size (value of the *size* property) of the GridElement.

2.3.3.6 Public void setSize (Dimension newSize)

Sets the size of the GridElement. (Sets the *size* property to newSize)

2.3.3.7 Public boolean isNumeric ()

Returns information about whether the GridElement contents are numeric (returns the value of the *numeric* property)

2.3.3.8 Public void setNumeric (boolean newNumeric)

Sets the *numeric* property of the GridElement to newNumeric.

2.3.3.9 Public float getData ()

Returns the data contained in the GridElement (value of the *data* property).

2.3.3.10 Public void setData (float newData)

Sets the data contained in the GridElement. (sets the *data* property to newData).

2.3.3.11 Public String getDisplayText ()

Returns the text displayed by the GridElement (value of the *displayText* property).

2.3.3.12 Public void setDisplayText (String newDisplayText)

Sets the text displayed by the GridElement (sets the *displayText* property to newDisplayText).

2.3.3.13 Public int getFontSize ()

Returns the font size of the GridElement (value of the fontSize property).

2.3.3.14 Public void setFontSize (int newFontSize)

Sets the font size of the GridElement. (Sets the *fontSize* property to newFontSize).

2.3.3.15 Public int getFontWeight ()

Returns the font weight of the GridElement. (value of the *fontWeight*).

2.3.3.16 Public void setFontWeight (int newFontWeight)

Sets the font weight of the GridElement. (sets the *fontWeight* property to *newFontWeight*).

2.3.3.17 Public boolean isBold ()

Returns information about whether the text in the GridElement is displayed in bold letters. (value of the *bold* property).

2.3.3.18 Public void setBold (boolean newBold)

Sets the *bold* property of the GridElement to newBold.

2.3.3.19 Public boolean isItalicized ()

Returns information about whether the text in the GridElement is italicized. (value of the *italicized* property).

2.3.3.20 Public boolean setItalicized (boolean newItalicized)

Sets the *italicized* property of the GridElement to newItalicized.

2.3.3.21 Public Color getColor ()

Returns the current color of the text displayed in the GridElement (value of the *color* prperty)

2.3.3.22 Public void setColor (Color newColor)

Sets the color of the GridElement (sets the *color* property to newColor).

2.3.3.23 Public void displayBalloon (Balloon newBalloon)

Displays newBalloon at the GridElement's lower right corner.

2.3.4. Events

2.4. ActiveTable
2.5. Text Area
2.6. Buttons

An ActiveTable can have one or more analyses associated with it. These are the entities that control the nature of the pattern displayed to the user for FastPattern implementation. They are represented using the **AttributeCluster** class.

Issues in Data Mining on the Internet

The following issues are considered in this document :

1. Consumer vs. Corporate user

This is more of a "thin client" vs. "Fat Client" issue.
"Thin Client" : as memory constraints are likely to exist, it would be preferable to mine the data on the remote server, and bring the results across to the client.
"Fat Client" : it would be useful to take advantage of the computational power of the client, and mine the data on the client.

Scenario	Pros	Cons
Data Mining on the client	1. No special server software required to process Data Mining requests 2. Web / Database Server is not loaded 3. No extra traffic on the network	1. Client may take longer to show Data Mining results
Data Mining on the server	1. Special server software required 2. Extra load on Web / Database server 3. Server software repeats work for same Data Mining queries from different clients 4. Extra traffic on network, to transmit results	1. Server is scaleable, and hence, *might* be faster

Note : It may not always be possible for a designer to tell beforehand whether the client is likely to be "thin" or "fat". On the Internet, this is practically impossible. On a corporate Intranet, however, this might be possible.

2. Runtime (user) vs. Design-time (developer) mining

Design-time Mining : Using Java Beans, it is possible to save the state of various objects at the time of design. This implies that when a developer designs an ActiveTable™, she can specify the Data Mining queries, and these could be saved along with the object. When the client requests the object, the results of the queries are sent along with the object.
Runtime Mining : Only the data mining code and data are sent to the client, and the actual mining is performed there.

Scenario	**Pros**	**Cons**
Design-time Mining	1. User does not have to wait for mining results 2. The (presumably higher) computational power of the server can be used.	1. Extra traffic on the network (as the Data Mining results have to be sent along with the applet) 2. The results sent to the client may not ever be seen by the user
Runtime Mining	1. No extra load on the network (as the client does her own mining) 2. Client mines data only when required, so all results are always seen by the user	1. User may have a long wait for mining results (especially for "thin" clients)

3. Small tables vs. large tables

The "smallness" of a table, with regard to its amenability for Data Mining, is determined by the following factors:

a. The number of attributes selected by the user for data mining – this can be limited to 3 or 4
b. The number of unique values for each attribute chosen
c. The depth of mining required – this can be limited
d. The number of records to be mined.

Mining is faster on small tables than on large tables.
The table "smallness", more than any of the above factors, should guide the decision of where the data mining is performed. Consider :

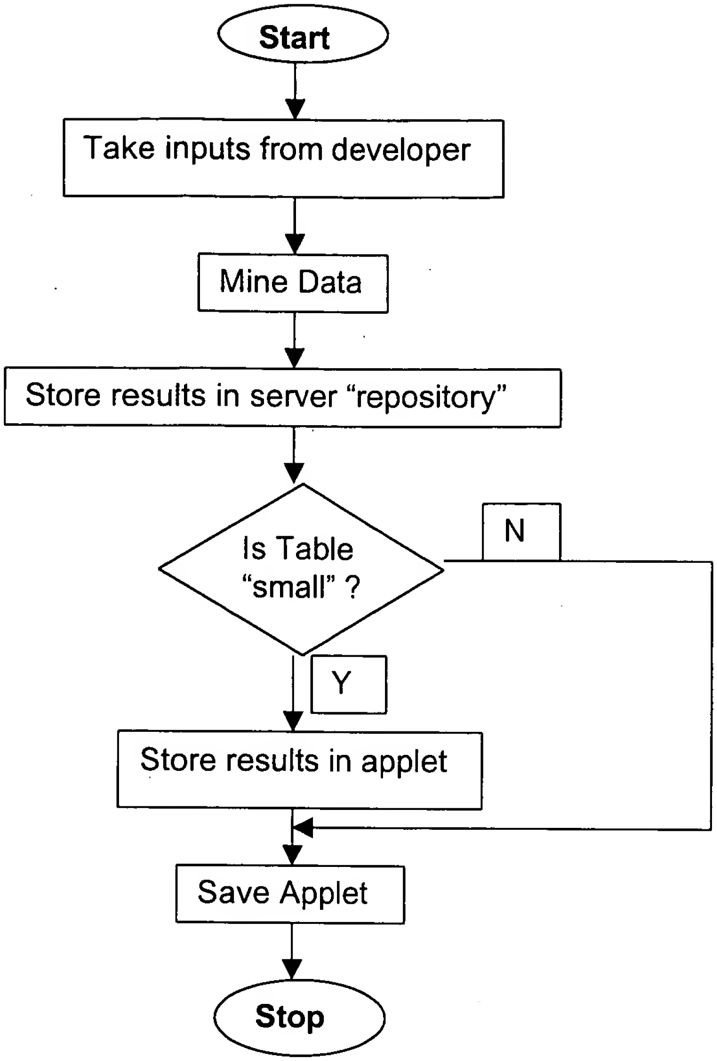
a. It is not always possible to determine whether a potential client is "thin" or "fat". However, it is **always** possible to determine the "smallness" of a table, at the time when the ActiveTable™ is being designed.
b. The ability of the client to mine data in as short a time as possible is obviously related to the "smallness" of the table being mined. If the table is small enough, even an "ultra-thin" client may be able to mine satisfactorily, and if the table is huge, even a big server may take minutes to mine it.

4. ActiveX vs. Java
We are dependent on the Java-to-ActiveX Bridge to convert code from one model to the other. This has to be tested thoroughly to ensure that the conversion is accurate. If necssary, code modifications may have to be made.

An alternative design paradigm

The design methodology suggested below avoids some of the disadvantages of mining data purely on the server, or purely on the client.
The design algorithm would be as follows :

1. Based on the table size, and the developer inputs (when the ActiveTable™ is being designed), decide whether the table is "small" or "big".
2. For both types of tables, mine the data on the server, during the design phase. (The possibility of batch-processing several ActiveTable™ designs, so as to obviate the need for the designer's presence during the mining can be investigated).
3. For both types of tables, store the Data Mining results on a "repository" on the server. This "repository" is an intelligent entity, that can respond to RMI (Remote Method Invocations) calls by clients, responding with Data Mining results, as and when required.
4. For "small" tables, store the results along with the applet also. If, for some reason, the client is not able to load all the results, the "repository" would serve as a backup.
5. For "big" tables, do not store the results on the client applet.
6. When the applet is sent to the client, and the user requests some mining results, the applet first checks its local store. If the result is found, it is displayed to the user. If it is not, the client invokes a method on the remote "repository" entity, and displays the results returned.



Design Procedure

This method has significant advantages :

1. For small tables, access to Data Mining results will be extremely fast.
2. For big tables, the data have already been mined. Only the results are requested from the server. Consequently, network load is reduced, and access time is lower.
3. For big tables, the data mining results are sent to the client only when the user asks for them. Consequently, the problem of sending extra information to each client is avoided. For small tables, all the results are still sent to the client, whether they are displayed to the user or not, but this is probably less of a problem, due to the "smallness" of the table.
4. The server will never be faced with a scenario where it is mining the same data several times to answer the **same** question from several clients. Since the results are stored, it will merely have to fetch the results and transmit them to the client.
5. It is possible to cache results on the client, even for big tables, so that repeated requests do not have to be made to the server. This speeds up access even further.

Architecture Design of InterScope™

Document ID :
Owner :

Virtual Gold Confidential

1. Introduction

InterScope is a tool aimed at web browser users to enable them to mine tabular data on an intranet / Internet. It would enable them to discover interesting patterns in the data, with minimal inputs.

InterScope™ aims at fulfilling the following requirements :

(a) Basic table functionality on the web – the user should be able to represent the data from any data source using the InterScope™ ActiveTable™. The table should have the same look and feel as a normal HTML table.
(b) FastPatterns – when the user moves her mouse on a column of the ActiveTable™, she should be able to view some interesting facts about the column.
(c) Data Mining, using secondary data sources – on the click of a button, the user should be able to mine data from several data sources, and link in information from other data sources (such as video)
(d) User Question Support – At the click of a button, the user should be able to frame and ask her own question, and get answers from different data sources, also linking her to secondary sources of information (such as text files, or video clips)

Section 2 describes the InterScope™ design goals. Section 3 describes the InterScope™ architecture.

2. Design Goals

The InterScope™ design is aimed at satisfying the following constraints :

2.1. Customizability

Reusability of InterScope™ components was a primary design goal. An InterScope™ table will be designed by a web developer, who should be able to mix-and-match various InterScope™ components together to create a solution that best fits her needs. For example, a developer might decide to include only basic ActiveTable™ functionality on her web page, without any data mining / user question support. On the other hand, she could choose to include FastPattern, data mining and User Question support along with the basic ActiveTable™. She should be able to build solutions to meet both these scenarios, as well as various options in between, using standard Builder tools.

2.2. Modularity

A primary design goal was to provide support for the requirements mentioned in Section 1 in a phased manner. The requirements are to be addressed in the order given above. Hence, the design should be modular to allow for easy addition of new features.

2.3. Flexibility

InterScope™ is to be used on a wide variety of platforms on Intranets / the Internet. At the time of design, it is not possible to predict the nature of the end-user environment. The end-user could be working on platforms ranging from embedded devices (such as a pager) and "thin" clients or network computers, to powerful desktop machines and workstations. This environment could have major effects on the performance of the InterScope™ table, especially when computation-intensive data mining queries are being run.

The design should be flexible, such that the when the ActiveTable™ is being designed by the Internet / Intranet developer, portions (or all) of the data and code can reside on the web server, and be brought to the client, as and when required. The design should support this distributed functionality.

3. InterScope™ Architecture

The main component of InterScope™ is the ActiveTable™. This is an "active" representation of tabular data, in which each tabular element can be made to respond to user actions (such as mouse movements).

3.1. The ActiveTable™

The ActiveTable™ is the basic InterScope™ component. It is always run on the client.

3.1.1. ActiveTable™ Properties

An ActiveTable™ has various properties, such as the Data Source, text color, font size, etc., all of which the web developer can specify while designing the ActiveTable™. (The developer will be assisted by an ActiveTable™ customizer wizard while designing the ActiveTable™). Except for the Data Source property, all of the others will have standard default values.

3.1.2. ActiveTable™ Elements

An ActiveTable™ has the following elements :

3.1.2.1 TitleGridArray

A single-row GridArray, consisting of a number of GridElements, for representing the titles of the various columns. This is always sent to the client. The title text properties of the ActiveTable™ are reflected by the TitleGridArray.

3.1.2.2 DataGridArray

A GridArray representing the actual data. This is always sent to the client. The normal text properties of the ActiveTable™ are reflected by the DataGridArray. The DataGridArray may have scroll bars to aid the user in viewing large tables. In addition, it has an array of ColumnManager elements, each of which contains :

(1) Information about the totals of various "singles" in a column.
(2) Information relating to the categorization of the column

3.2. The ClusterSet

The developer can add one or more Clusters to a basic ActiveTable™ to induce FastPattern and Data Mining functionality. A cluster is thus a JavaBean, that the user can add to an ActiveTable™. These clusters are stored in a ClusterSet. (As in the case of the basic ActiveTable™, the developer would be assisted by a wizard when she designs clusters).

Each cluster will contain the following information :
(1) Information about the attributes in the cluster
(2) Information about the "singles" in the cluster
(3) FastPattern information about the cluster

The ClusterSet could exist only on the server or on both the client and the server, depending on the size of the ActiveTable™ data. If the data are small, the ClusterSet is stored on the server, as well as attached to the ActiveTable™ applet that would be sent to the client at runtime. If the table is large, it is stored only on the server. This decision is (automatically) taken at the time the ActiveTable™ is designed, transparent to the developer, according to some criterion, still to be developed.

In either case, code is added to the ActiveTable™ to allow for remote invocation of the ClusterSet's methods.

3.2.1. ClusterSet Component Elements

3.2.1.1 FastPatternBalloon

The FastPatternBalloon component is added automatically to the ActiveTable™ when the developer adds a cluster to the ActiveTable™. This is activated when the user's mouse is on a column of the ActiveTable™ that belongs to a cluster that the developer has defined, and displays a FastPattern (some interesting information relating the column to other attributes in the cluster).

The FastPattern balloon is always sent to the client along with the ActiveTable™.

3.3. The DataMiner Component

The developer can add a DataMiner Component to the ActiveTable™ to enable data mining across different data sources. The DataMiner component customizer wizard would collect information about the secondary data sources, etc. from the developer. At least one cluster should have been created before the DataMiner component can be added, as the mining is performed on the attributes in a luster. (If a cluster has not been created, the developer will be prompted to create one, by the DataMiner customizer).

The DataMiner component contains the code (Attribute Focusing) for data mining. It is treated similar to the ClusterSet (retained only on the server for large tables, and kept on both client and server for small data sources).

3.4. The YourQuestion Component

The developer can add this component to an ActiveTable™ to enable a user to ask her own questions of the data. If other data sources have been added, the user will be able to ask questions across different sources. It is treated similar to the ClusterSet (retained only on the server for large tables, and kept on both client and server for small data sources).

3.4.1. YourQuestion Component Elements

3.4.1.1 Index

The index is a means of fast access to the data in the DataGridArray and aids in answering user questions. It is added to the ActiveTable™ when the YourQuestion component is added to it.

3.5. *Design Scenarios*

The following table depicts various possibilities for mixing and matching the InterScope™ components :

Configuration	Functionality
ActiveTable™	Basic table functionality
ActiveTable™ + ClusterSet	Basic table functionality + FastPatterns
ActiveTable™ + ClusterSet + DataMiner	Basic table functionality + FastPatterns + Data Mining capability across data sources
ActiveTable™ + ClusterSet + YourQuestion	Basic table functionality + FastPatterns + User question capability across data sources
ActiveTable™ + ClusterSet + DataMiner + YourQuestion	Basic table functionality + FastPatterns + Data Mining capability across data sources + User question capability across data sources